[R-bloggers] How to Prepare Data (and 6 more aRticles)

Expéditeur : R-bloggers (noreply+feedproxy@google.com)

À : k.alahassa123@yahoo.fr

Date : jeudi 26 septembre 2019 à 23:13 UTC

[R-bloggers] How to Prepare Data (and 6 more aRticles)

- **How to Prepare Data**
- **Multi-Armed Bandits as an A/B Testing Solution**
- **Debuting in a VFL/AFL Grand Final is rare**
- **Multiple imputation support in Finalfit**
- **EARL London review**
- **How to build analytics platforms – Part 5: Data visualization and reliable results**
- **Time series forecasting with random forest**

How to Prepare Data

Posted: 26 Sep 2019 09:46 AM PDT

[This article was first published on **R – Win-Vector Blog**, and kindly contributed to **R-bloggers**]. (You can report issue about the content on this page **here**)

Want to share your content on R-bloggers? **click here** if you have a blog, or **here** if you don't.

Real world data can present a number of challenges to data science workflows. Even properly structured data (each interesting measurement already landed in distinct columns), can present problems, such as missing values and high cardinality categorical variables.

In this note we describe some great tools for working with such data.

For an example: consider the **KDD 2009 contest data**. Though this data structured, it is not immediately compatible with a number of high-quality machine learning packages (such as xgboost). As we see in the following **Python extract**, xgboost raises an exception on this data due to the issues we raised above (non-numeric column types, and also missing values):

```
fitter = xgboost.XGBClassifier(n_estimators=10, max_depth=3, objective='binary:logistic')
try:
    fitter.fit(d_train, churn_train)
except Exception as ex:
    print(ex)

# DataFrame.dtypes for data must be int, float or bool.
#                Did not expect the data types in fields Var191, Var192, Var193, ...
```

vtreat is a family of packages (**in R** and **in Python**) to prepare structured data for machine learning or data science projects in a statistically sound manner. The goal of vtreat is to transform arbitrary structured data into “clean” pure numeric data. This “clean” data has no missing values, and retains most of the information relating explanatory variables to the dependent variable to be predicted.

The vtreat principles include:

- do a very good job
- work fast, and at production scale
- minimize interference: leave as many opportunities open to the user and downstream modeling software as possible.

The last point (minimizing interference / maximizing opportunity) is a subtle, but important one. vtreat does not choose any one language (it is currently available both in R or Python, leaving the choice of working in R or Python to the user) and tries to be low to moderate dependency (for instance not bringing in a deep learning system, thus leaving the choice of such systems for later steps again open).

The overall intent is that by automating the domain independent steps in data preparation we leave the analyst with much more time to work on even more critical domain dependent steps.

In all cases, designing a vtreat transform should be a one-liner. Later application of the transform should also be a one-liner (the “one line” is `prepare()` in R, and `.transform()` in Python).

To trust “one liners” one needs a good discussion of the theory behind them (both for learning and to cite), and worked examples.

The vtreat theory can be found here: . This helps you both learn how the vtreat transforms work, and also is itself a quick way to document them when used in your own work (such as in a “methods” section).

And we have a growing organized family of documentation and simple examples organized by task here:

- **Regression**: **`R` regression example**, **`Python` regression example**.
- **Classification**: **`R` classification example**, **`Python` classification example**.

- **Unsupervised tasks**: `R` **unsupervised example**, `Python` **unsupervised example**.
- **Multinomial classification**: `R` **multinomial classification example**, `Python` **multinomial classification example**.

Overall vtreat R documentation (including how to install) can be found **here**, and vtreat Python documentation (including how to install) **here**.

As **we have said before**: if you aren't using something like vtreat in your data science projects: you are really missing out (and making more work for yourself).

We really hope you try vtreat for one of your projects. We think you will have a great experience.

To **leave a comment** for the author, please follow the link and comment on their blog: **R – Win-Vector Blog**.

R-bloggers.com offers **daily e-mail updates** about **R** news and tutorials about **learning R** and many other topics. **Click here if you're looking to post or find an R/data-science job**.

Want to share your content on R-bloggers? **click here** if you have a blog, or **here** if you don't.

This posting includes an audio/video/photo media file: Download Now

Multi-Armed Bandits as an A/B Testing Solution

Posted: 26 Sep 2019 05:00 AM PDT

[This article was first published on **INWT-Blog-RBloggers**, and kindly contributed to **R-bloggers**]. (You can report issue about the content on this page **here**)

Want to share your content on R-bloggers? **click here** if you have a blog, or **here** if you don't.

These days, most people are familiar with the concept of A/B testing. This is one of the most common ways to make advertising decisions, particularly in online marketing. In an A/B test, the customer base is divided into two or more groups, each of which is served a different version of whatever is being tested (such as a special offer, or the layout of an advertising campaign). At the end of the test, whichever variant was most successful is pursued for the customer base at large.

While this method is tried and true, there are some potential situations where it is less than ideal. For example, it is important for the integrity of the experiment that the results of A/B tests not be checked while the program is ongoing. But what if it turns out that one of the variants being tested is performing very poorly, or extremely well? That means that a portion of the customers aren't going to be served optimally.

If there is only one test, this opportunity cost is probably manageable, and worth the investment to discover the best possible variant. But in the event that many variants are being tested, a greater proportion of the customers will be served a suboptimal variant for a longer period of time, potentially resulting in significant costs.

In an ideal world, we'd want to be able to rule out the variants that are performing poorly very quickly, and only conduct a full A/B test on those variants that have promise or have proven to perform well.

Multi-Armed Bandits as a Solution

Originally, slot machines were operated by pulling a lever, earning the name "one-armed bandits" given their penchant for cleaning out players' bank accounts. The name "Multi-Armed Bandit" comes from the idea of playing multiple slot machines at once, each with a different unknown payoff rate. Our goal as the player would be to figure out which machine has the highest payoff rate, and play that machine more often than the others to maximize our rewards.

While it might seem like our goal would be to exclusively play the higher-payoff machine, unless we know for certain what the exact payoff rates are in advance (which we can't in this scenario), we would want to continue playing the other machines for as long as it takes to be reasonably certain that we have selected the correct machine to prioritize.

This brings us to the concept of the exploration vs. exploitation trade-off (in layman's terms, "learn or earn"). Basically, this is the trade-off between reward later (exploration) or now (exploitation).

An A/B test is purely exploratory: customers are randomly assigned to a variant, and the achievable payoff is simply the average payoff of all the variants (meaning that, by design, it will always be lower than the payoff of the strongest variant). There is no conscious exploitation of a higher-earning variant, which leads to "regret": the difference between the optimal policy that we would choose if we knew the true payoff, and the action taken.

Bandit algorithms, on the other hand, seek to strike a balance between exploration and exploitation by exploring each variant sufficiently enough to identify the strongest variant, and then primarily exploiting the optimal action to maximize the total reward. The algorithm also continues to explore other possible variants to discover whether they become stronger in the future. The idea is to end up with a higher final reward (and thus, less regret), while still allowing the algorithm to continue gathering data on other variants.

Varieties of Multi-Armed Bandit Algorithms

There are many kinds of Bandit algorithms. Three of the most common – Epsilon Greedy, Upper Confidence Bounds, and Thompson Sampling – are briefly discussed here, to illustrate how these algorithms work in practice.

Epsilon Greedy

In the Epsilon Greedy (or ε-greedy) algorithm, after an initial number of purely exploratory trials, a random lever is pulled a fraction ε of the time. The rest of the time (1 – ε), the lever with the highest known payoff is pulled. For example, if we set ε to 0.10, 10% of the time the algorithm will explore random alternatives, and 90% of the time it will exploit the variant that is performing the best. We can expand on the ε-greedy algorithm by having it reduce the value of ε over time, thus limiting the possibility of continuing to explore once we're aware of each lever's payoff. This is referred to as Decayed Epsilon Greedy.

Upper Confidence Bounds

The Upper Confidence Bounds (UCB) family of algorithms are based on the principle of optimism in the face of uncertainty, which assumes that the unknown payoff of each lever is as high as the observable data indicate is possible. UCB initially explores randomly to try to get an understanding of the true probability of a lever being successful. Then, the algorithm adds an "exploration bonus" to levers it is unsure about, prioritizing those until it becomes more sure about the lever's performance. This exploration bonus decreases over each round, similar to the Decayed Epsilon Greedy.

Thompson Sampling

It is also possible that a winning lever may initially appear weak, and thus not be explored by a "greedy" algorithm sufficiently enough to determine its true payoff. Thompson Sampling is a Bayesian, non-greedy alternative. In this algorithm, a probability distribution of the true success rate is built for each variant based on results that have already been observed. For each new trial, the algorithm chooses the variant based on a sample of its posterior probability of having the best payout. The algorithm learns from this, ultimately bringing the sampled success rates closer to the true rate.

Simulation

To illustrate, let's use a simplified example to compare a more traditional A/B test to Epsilon Greedy and Thompson Sampling. In this scenario, a customer can be shown one of five variants of an advertisement. For our purposes, we will assume that Ad 1 performs the worst, with a 5% conversion rate. Each ad performs 5% better than the last, with the best performer being Ad 5, at 25% conversion. We'll do 1,000 trials, which means that in an idealized, hypothetical world, the number of conversions we could get by only showing the optimal ad would be 250 (given a 25% conversion rate over 1,000 trials).

In our traditional A/B test, each ad is initially served an equal number of times. After a set period (halfway through the total 1,000 trials, in our case), the highest performer(s) are isolated by performing a statistical significance test and removing ads that are significantly worse than the top performer, and from then on only those are shown.

```
 # A/B TEST # setup  horizon <- 1000L simulations <- 1000L conversionProbabilities <- c(0.05, 0.10, 0.15, 0.20, 0.25)
nTestSample <- 0.5 * horizon clickProb <- rep(NA, simulations) adDistMatrix <- matrix(NA, nrow = simulations,
```

```
ncol = length(conversionProbabilities)) adDistMatrixAB <- matrix(NA, nrow = simulations,                        ncol =
length(conversionProbabilities)) # simulation for(i in 1:simulations){   testSample <- sapply(conversionProbabilities,
function(x) sample(0:1, nTestSample, replace = TRUE, prob = c(1 - x, x)))      testColumns <-
(1:length(conversionProbabilities))[-which.max(colSums(testSample))]      p.values <- sapply(testColumns, function(x)
prop.test(x = colSums(testSample[, c(x, which.max(colSums(testSample)))]),
n = rep(nTestSample, 2))$p.value)      adsAfterABTest <- (1:length(conversionProbabilities))[-
testColumns[which(p.values < 0.05)]]      # now just with the best performing ad(s)   ABSample <-
sapply(conversionProbabilities[adsAfterABTest],                        function(x) sample(0:1,
round((horizon - nTestSample) * length(conversionProbabilities) /
length(adsAfterABTest), 0),                                                replace = TRUE, prob = c(1 - x, x)))
clickProbTest <- sum(as.vector(testSample)) / length(unlist(testSample))   clickProbAB <- sum(as.vector(ABSample)) /
length(unlist(ABSample))      clickProb[i] <- clickProbTest * (nTestSample / horizon) + clickProbAB * (1 - nTestSample
/ horizon)      # distribution of which ads were seen over the course of all trials   adDistMatrix[i,] <- rep(1 /
length(conversionProbabilities), length(conversionProbabilities))   adDistributionAB <- rep(0,
length(conversionProbabilities))   adDistributionAB[adsAfterABTest] <- rep(1 / length(adsAfterABTest),
length(adsAfterABTest))   adDistMatrixAB[i,] <- adDistributionAB    } # total payoff ABPayoff <- (nTestSample *
clickProbTest) + (nTestSample * clickProbAB)
```

The A/B test resulted in a total payoff of 187.4 conversions over 1,000 trials, which is about 75% of the total possible payoff. We can also visualize which ads were seen to get a better feel for the process.

As expected, during the first 500 trials the ads were seen an even proportion of the time. In the next 500 trials, Ad 5 (the highest performer) was seen the most often, followed by Ad 4, and very rarely, Ad3.

In the next simulation, we can use the contextual package to look at an Epsilon Greedy algorithm with ε = 0.10. This means that it will explore randomly only 10% of the time, exploiting 90% of the time.

```
library(contextual) # EPSILON GREEDY horizon <- 1000L simulations <- 1000L conversionProbabilities <- c(0.05, 0.10,
0.15, 0.20, 0.25)  bandit <- BasicBernoulliBandit$new(weights = conversionProbabilities) policy <-
EpsilonGreedyPolicy$new(epsilon = 0.10) agent <- Agent$new(policy, bandit) historyEG <- Simulator$new(agent, horizon,
simulations)$run() plot(historyEG, type = "arms",       legend_labels = c('Ad 1', 'Ad 2', 'Ad 3', 'Ad 4', 'Ad 5'),
legend_title = 'Epsilon Greedy',      legend_position = "topright",      smooth = TRUE) summary(historyEG)
```

Here we can see that the Epsilon Greedy algorithm is quickly able to determine that Ad 5 is superior, improving over time. The total payoff is 217.11 conversions, or about 87% of the total possible payoff. This is also approximately 16% higher than with the A/B test, because the algorithm is able to begin exploiting the most successful ad much more quickly than with traditional methods.

Finally, we can compare both of these examples to Thompson Sampling, also using the contextual package.

```
# THOMPSON SAMPLING horizon <- 1000L simulations <- 1000L conversionProbabilities <- c(0.05, 0.10, 0.15, 0.20, 0.25)
bandit <- BasicBernoulliBandit$new(weights = conversionProbabilities) policy <- ThompsonSamplingPolicy$new(alpha = 1,
beta = 1) agent <- Agent$new(policy, bandit) historyThompson <- Simulator$new(agent, horizon, simulations)$run()
plot(historyThompson, type = "arms",       legend_labels = c('Ad 1', 'Ad 2', 'Ad 3', 'Ad 4', 'Ad 5'),
legend_title = 'Thompson Sampling',      legend_position = "topright",      smooth = TRUE) summary(historyThompson)
```

Thompson Sampling turns out to be our most successful model, resulting in a payoff of 219.97 (or almost 88% of the total conversions hypothetically achievable), over 17% more than with traditional A/B testing.

What's more, while these results appear not too different from those in the Epsilon Greedy example, it is important to note that this would change with further trials. Since Epsilon Greedy is committed to exploring 10% of the time throughout the entire experiment, there is a limit to the extent that the algorithm can continue to optimize. Thompson Sampling is not similarly constrained, and rather can fully exploit the most successful option once it has been determined.

Overall, Bandit algorithms can provide an excellent solution for many use cases. In essence, they are a "light" form of reinforcement learning, where the algorithm auto-optimizes without knowledge of the task in advance, and earns rewards from the environment. Reinforcement learning, and its applications for online marketing, will be discussed in a future blog.

To **leave a comment** for the author, please follow the link and comment on their blog: **INWT-Blog-RBloggers**.

R-bloggers.com offers **daily e-mail updates** about **R** news and tutorials about **learning R** and many other topics. **Click here if you're looking to post or find an R/data-science job**.

Want to share your content on R-bloggers? **click here** if you have a blog, or **here** if you don't.

This posting includes an audio/video/photo media file: **Download Now**

Debuting in a VFL/AFL Grand Final is rare

Posted: 26 Sep 2019 04:43 AM PDT

[This article was first published on **R – What You're Doing Is Rather Desperate**, and kindly contributed to **R-bloggers**]. (You can report issue about the content on this page **here**)

Want to share your content on R-bloggers? **click here** if you have a blog, or **here** if you don't.

When Marlion Pickett **runs onto the M.C.G for Richmond** in the AFL Grand Final this Saturday, he'll be only the sixth player in 124 finals to debut on the big day.

The sole purpose of this blog post is to illustrate how incredibly easy it is to figure this out, thanks to the **dplyr** and **fitzRoy** packages.

```
library(dplyr)
library(fitzRoy)

afldata <- get_afltables_stats()

afldata %>%
  select(Season, Round, Date, ID, First.name, Surname, Playing.for,
         Home.team, Home.score, Away.team, Away.score) %>%
  group_by(ID) %>%
  arrange(Date) %>%
  # a player's first game
  slice(1) %>%
  ungroup() %>%
  # grand finals only
  filter(Round == "GF") %>%
  # get the winning/losing margin
  mutate(Margin = case_when(Playing.for == Home.team ~ Home.score - Away.score,
                            TRUE ~ Away.score - Home.score)) %>%
  select(-Home.team, -Away.team, Home.score, Away.score)
```

Season	Round	Date	ID	First.name	Surname	Playing.for	Margin
1908	GF	1908-09-26	5573	Harry	Prout	Essendon	-9
1920	GF	1920-10-02	6677	Billy	James	Richmond	17
1923	GF	1923-10-20	6915	George	Rawle	Essendon	17
1926	GF	1926-10-09	3824	Francis	Vine	Melbourne	57

Season	Round	Date	ID	First.name	Surname	Playing.for	Margin
1952	GF	1952-09-27	9361	Keith	Batchelor	Collingwood	-46

To **leave a comment** for the author, please follow the link and comment on their blog: **R – What You're Doing Is Rather Desperate**.

R-bloggers.com offers **daily e-mail updates** about **R** news and tutorials about **learning R** and many other topics. **Click here if you're looking to post or find an R/data-science job**.

Want to share your content on R-bloggers? **click here** if you have a blog, or **here** if you don't.

This posting includes an audio/video/photo media file: **Download Now**

Multiple imputation support in Finalfit

Posted: 25 Sep 2019 12:05 PM PDT

[This article was first published on **R – DataSurg**, and kindly contributed to **R-bloggers**]. (You can report issue about the content on this page **here**)

Want to share your content on R-bloggers? **click here** if you have a blog, or **here** if you don't.

We are using multiple imputation more frequently to “fill in” missing data in clinical datasets. Multiple datasets are created, models run, and results pooled so conclusions can be drawn.

We’ve put some improvements into Finalfit on GitHub to make it easier to use with the `mice` package. These will go to CRAN soon but not immediately.

See **finalfit.org/missing.html** for more on handling missing data.

Let’s get straight to it by imputing smoking status in a cancer dataset.

Install

```
devtools::install_github("ewenharrison/finalfit")
library(finalfit)
library(dplyr)
```

Create missing data for example

```
# Smoking missing completely at random

set.seed(1)

colon_s = colon_s %>%
  mutate(
    smoking_mcar = sample(c("Smoker", "Non-smoker", NA),
      dim(colon_s)[1], replace=TRUE,
      prob = c(0.2, 0.7, 0.1)) %>%
    factor() %>%
    ff_label("Smoking (MCAR)")
    )

# Smoking missing conditional on patient sex
colon_s$smoking_mar[colon_s$sex.factor == "Female"] =
  sample(c("Smoker", "Non-smoker", NA),
    sum(colon_s$sex.factor == "Female"),
    replace = TRUE,
    prob = c(0.1, 0.5, 0.4)
  )

colon_s$smoking_mar[colon_s$sex.factor == "Male"] =
  sample(c("Smoker", "Non-smoker", NA),
    sum(colon_s$sex.factor == "Male"),
    replace=TRUE, prob = c(0.15, 0.75, 0.1)
  )

colon_s = colon_s %>%
  mutate(
    smoking_mar = factor(smoking_mar) %>%
    ff_label("Smoking (MAR)")
  )
```

Check data

```
explanatory = c("age", "sex.factor",
  "nodes", "obstruct.factor",
```

```
  "smoking_mcar", "smoking_mar")
 dependent = "mort_5yr"
 colon_s %>%
  ff_glimpse(dependent, explanatory)

 Continuous
           label var_type   n missing_n missing_percent mean   sd  min quartile_25 median quartile_75  max
age   Age (years)     929         0             0.0 59.8 11.9 18.0        53.0   61.0        69.0 85.0
nodes       nodes     911        18             1.9  3.7  3.6  0.0         1.0    2.0         5.0 33.0

Categorical
                         label var_type   n missing_n missing_percent levels_n
sex.factor                 Sex     929         0             0.0        2
obstruct.factor      Obstruction     908        21             2.3        2
mort_5yr        Mortality 5 year     915        14             1.5        2
smoking_mcar      Smoking (MCAR)     828       101            10.9        2
smoking_mar        Smoking (MAR)     719       210            22.6        2
                                       levels  levels_count   levels_percent
sex.factor                        "Female", "Male"      445, 484           48, 52
obstruct.factor            "No", "Yes", "(Missing)"  732, 176, 21 78.8, 18.9,  2.3
mort_5yr              "Alive", "Died", "(Missing)"  511, 404, 14 55.0, 43.5,  1.5
smoking_mcar    "Non-smoker", "Smoker", "(Missing)" 645, 183, 101       69, 20, 11
smoking_mar     "Non-smoker", "Smoker", "(Missing)" 591, 128, 210       64, 14, 23
```

Multivariate Imputation by Chained Equations (mice)

`mice`is a great package and contains lots of useful functions for diagnosing and working with missing data. The purpose here is to demonstrate how `mice` can be integrated into the Finalfit workflow with inclusion of model from imputed datasets in tables and plots.

Choose variables to impute and variables to impute from

`finalfit::missing_predictorMatrix()`makes it easy to specify which variables do what. For instance, we often do not want to impute our outcome or explanatory variable of interest (exposure), but do want to use them to impute other variables.

This is straightforward to code using the arguments `drop_from_imputed` and `drop_from_imputer`.

```
library(mice)
```

```
# Specify model
explanatory = c("age", "sex.factor", "nodes",
  "obstruct.factor", "smoking_mar")
dependent = "mort_5yr"

# Choose not to impute missing values
# for explanatory variable of interest and
# outcome variable.
# But include in algorithm for imputation.
predM = colon_s %>%
        select(dependent, explanatory) %>%
        missing_predictorMatrix(
                drop_from_imputed = c("obstruct.factor", "mort_5yr")
        )
```

Create imputed datasets

A set of multiple imputed datasets (mids) can be created as below. Various checks should be performed to ensure you understand the data that has been created. See **here**.

```
mids = colon_s %>%
  select(dependent, explanatory) %>%
  mice(m = 4, predictorMatrix = predM)    # Usually m = 10
```

Run models

Here we sill use a logistic regression model. The `with.mids()` function takes a model with a formula object, so use base R functions rather than Finalfit wrappers.

```
fits = mids %>%
  with(glm(formula(ff_formula(dependent, explanatory)),
    family="binomial"))
```

We now have multiple models run with each of the imputed datasets. We haven't found good methods for combining common model metrics like AIC and c-statistic. I'd be interested to hear from anyone working on methods for this. Metrics can be extracted for each individual model to give an idea of goodness-of-fit and discrimination. We're not suggesting you use these to compare imputed datasets, but could use them to compare models containing different variables created using the imputed datasets, e.g.

```
fits %>%
  getfit() %>%
  purrr::map(AIC)
[[1]]
[1] 1192.57

[[2]]
[1] 1191.09

[[3]]
[1] 1195.49

[[4]]
[1] 1193.729

# C-statistic
fits %>%
  getfit() %>%
  purrr::map(~ pROC::roc(.x$y, .x$fitted)$auc)
[[1]]
Area under the curve: 0.6839

[[2]]
Area under the curve: 0.6818

[[3]]
Area under the curve: 0.6789

[[4]]
Area under the curve: 0.6836
```

Pool results

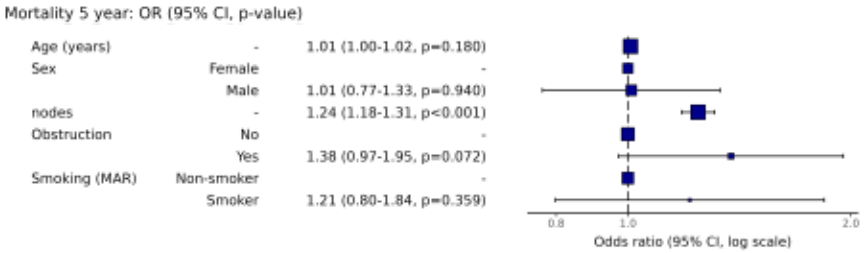
Rubin's rules are used to combine results of multiple models.

```
# Pool  results
fits_pool = fits %>%
  pool()
```

Plot results

Pooled results can be passed directly to Finalfit plotting functions.

```
# Can be passed to or_plot
colon_s %>%
  or_plot(dependent, explanatory, glmfit = fits_pool, table_text_size=4)
```



Put results in table

The pooled result can be passed directly to `fit2df()` as can many common models such as `lm()`, `glm()`, `lmer()`, `glmer()`, `coxph()`, `crr()`, etc.

```
# Summarise and put in table
fit_imputed = fits_pool %>%
  fit2df(estimate_name = "OR (multiple imputation)", exp = TRUE)
fit_imputed
```

```
         explanatory  OR (multiple imputation)
1                age 1.01 (1.00-1.02, p=0.212)
2     sex.factorMale 1.01 (0.77-1.34, p=0.917)
3              nodes 1.24 (1.18-1.31, p<0.001)
4 obstruct.factorYes 1.34 (0.94-1.91, p=0.105)
5  smoking_marSmoker 1.28 (0.88-1.85, p=0.192)
```

Combine results with summary data

Any model passed through `fit2df()` can be combined with a summary table generated with `summary_factorlist()` and any number of other models.

```
# Imputed data alone
## Include missing data in summary table
```

```
colon_s %>%
  summary_factorlist(dependent, explanatory, na_include = TRUE, fit_id = TRUE) %>%
  ff_merge(fit_imputed, last_merge = TRUE)

           label     levels       Alive        Died  OR (multiple imputation)
1    Age (years)  Mean (SD) 59.8 (11.4) 59.9 (12.5) 1.01 (1.00-1.02, p=0.212)
6            Sex     Female  243 (55.6)  194 (44.4)                          -
7                      Male  268 (56.1)  210 (43.9) 1.01 (0.77-1.34, p=0.917)
2          nodes  Mean (SD)   2.7 (2.4)   4.9 (4.4) 1.24 (1.18-1.31, p<0.001)
4    Obstruction         No  408 (56.7)  312 (43.3)                          -
5                       Yes   89 (51.1)   85 (48.9) 1.34 (0.94-1.91, p=0.105)
3                   Missing   14 (66.7)    7 (33.3)                          -
9  Smoking (MAR) Non-smoker  328 (56.4)  254 (43.6)                          -
10                    Smoker   68 (53.5)   59 (46.5) 1.28 (0.88-1.85, p=0.192)
8                   Missing  115 (55.8)   91 (44.2)                          -
```

Combine results with other models

Models can be run separately, or using the `finalfit()`wrapper including the argument `keep_fit_it = TRUE`.

```
colon_s %>%
  finalfit(dependent, explanatory, keep_fit_id = TRUE) %>%
  ff_merge(fit_imputed, last_merge = TRUE)

  Dependent: Mortality 5 year                  Alive        Died          OR (univariable)        OR (multivariable)
OR (multiple imputation)
1                 Age (years)  Mean (SD) 59.8 (11.4) 59.9 (12.5) 1.00 (0.99-1.01, p=0.986) 1.02 (1.00-1.03, p=0.010)
1.01 (1.00-1.02, p=0.212)
5                         Sex     Female  243 (47.6)  194 (48.0)                         -                         -
-
6                                   Male  268 (52.4)  210 (52.0) 0.98 (0.76-1.27, p=0.889) 0.88 (0.64-1.23, p=0.461)
1.01 (0.77-1.34, p=0.917)
2                       nodes  Mean (SD)   2.7 (2.4)   4.9 (4.4) 1.24 (1.18-1.30, p<0.001) 1.25 (1.18-1.33, p<0.001)
1.24 (1.18-1.31, p<0.001)
3                 Obstruction         No  408 (82.1)  312 (78.6)                         -                         -
-
4                                    Yes   89 (17.9)   85 (21.4) 1.25 (0.90-1.74, p=0.189) 1.26 (0.85-1.88, p=0.252)
1.34 (0.94-1.91, p=0.105)
```

```
7               Smoking (MAR) Non-smoker  328 (82.8)  254 (81.2)                        -                         -
-
8                                   Smoker   68 (17.2)   59 (18.8) 1.12 (0.76-1.65, p=0.563) 1.25 (0.82-1.89, p=0.300)
1.28 (0.88-1.85, p=0.192)
```

Model missing explicitly in complete case models

A straightforward method of modelling missing cases is to make them explicit using the forcats function `fct_explicit_na()`.

```
library(forcats)
colon_s %>%
  mutate(
    smoking_mar = fct_explicit_na(smoking_mar)
  ) %>%
  finalfit(dependent, explanatory, keep_fit_id = TRUE) %>%
  ff_merge(fit_imputed, last_merge = TRUE)

  Dependent: Mortality 5 year                  Alive        Died          OR (univariable)        OR (multivariable)
OR (multiple imputation)
1                 Age (years)  Mean (SD) 59.8 (11.4) 59.9 (12.5) 1.00 (0.99-1.01, p=0.986) 1.01 (1.00-1.02, p=0.119)
1.01 (1.00-1.02, p=0.212)
5                         Sex     Female  243 (47.6)  194 (48.0)                        -                         -
-
6                                   Male  268 (52.4)  210 (52.0) 0.98 (0.76-1.27, p=0.889) 0.96 (0.72-1.30, p=0.809)
1.01 (0.77-1.34, p=0.917)
2                       nodes  Mean (SD)   2.7 (2.4)   4.9 (4.4) 1.24 (1.18-1.30, p<0.001) 1.25 (1.19-1.32, p<0.001)
1.24 (1.18-1.31, p<0.001)
3                 Obstruction         No  408 (82.1)  312 (78.6)                        -                         -
-
4                                    Yes   89 (17.9)   85 (21.4) 1.25 (0.90-1.74, p=0.189) 1.34 (0.94-1.91, p=0.102)
1.34 (0.94-1.91, p=0.105)
8               Smoking (MAR) Non-smoker  328 (64.2)  254 (62.9)                        -                         -
-
9                                   Smoker   68 (13.3)   59 (14.6) 1.12 (0.76-1.65, p=0.563) 1.24 (0.82-1.88, p=0.308)
1.28 (0.88-1.85, p=0.192)
7                             (Missing)  115 (22.5)   91 (22.5) 1.02 (0.74-1.41, p=0.895) 0.99 (0.69-1.41, p=0.943)
-
```

Export tables to PDF and Word

As described **elsewhere**, `knitr::kable()` can be used to export good looking tables.

Dependent: Mortality 5 year		Alive	Died	OR (univariable)	OR (multivariable)	OR (multiple imputation)
Age (years)	Mean (SD)	59.8 (11.4)	59.9 (12.5)	1.00 (0.99-1.01, p=0.986)	1.01 (1.00-1.02, p=0.119)	1.01 (0.99-1.02, p=0.268)
Sex	Female	243 (47.6)	194 (48.0)	-	-	-
	Male	268 (52.4)	210 (52.0)	0.98 (0.76-1.27, p=0.889)	0.96 (0.72-1.30, p=0.809)	1.02 (0.77-1.34, p=0.909)
nodes	Mean (SD)	2.7 (2.4)	4.9 (4.4)	1.24 (1.18-1.30, p<0.001)	1.25 (1.19-1.32, p<0.001)	1.24 (1.18-1.31, p<0.001)
Obstruction	No	408 (82.1)	312 (78.6)	-	-	-
	Yes	89 (17.9)	85 (21.4)	1.25 (0.90-1.74, p=0.189)	1.34 (0.94-1.91, p=0.102)	1.32 (0.93-1.87, p=0.117)
Smoking (MAR)	Non-smoker	328 (64.2)	254 (62.9)	-	-	-
	Smoker	68 (13.3)	59 (14.6)	1.12 (0.76-1.65, p=0.563)	1.24 (0.82-1.88, p=0.308)	1.28 (0.84-1.97, p=0.248)
	(Missing)	115 (22.5)	91 (22.5)	1.02 (0.74-1.41, p=0.895)	0.99 (0.69-1.41, p=0.943)	-

To **leave a comment** for the author, please follow the link and comment on their blog: **R – DataSurg**.

R-bloggers.com offers **daily e-mail updates** about **R** news and tutorials about **learning R** and many other topics. **Click here if you're looking to post or find an R/data-science job**.

Want to share your content on R-bloggers? **click here** if you have a blog, or **here** if you don't.

This posting includes an audio/video/photo media file: Download Now

EARL London review

Posted: 25 Sep 2019 07:46 AM PDT

[This article was first published on **RBlog – Mango Solutions**, and kindly contributed to **R-bloggers**]. (You can report issue about the content on this page **here**)

Want to share your content on R-bloggers? **click here** if you have a blog, or **here** if you don't.

We hope you enjoyed EARL London as much as we did! We're just putting the finishing touches on our highlights recap, but until then you can view the presentations we have available **here**, and all the photos **here.**

We're so proud of the incredible speakers at this year's EARL Conference – from tackling human trafficking, to benefitting the NHS, it's always incredible to hear about the many different ways data science can be used for good.

Let's take a look at some of the feedback and comments we received on this year's London EARL:

From social:

*Great to have been on the same speaking bill as experts from **@BMW** **@sainsburys** the BBC and others talking about **#opensource**, **#R** and **#bigdata**. World class speakers on cutting edge topics to an international audience. Thanks **@earlconf**– see you next year!*

*Had a brilliant day today at my 3rd **@earlconf** Have come away with a million ideas, in particular on workflows and productionising R code, and it was brilliant to both see **@juliasilge** speak and then have her sit in on my presentation! Thanks **@MangoTheCat**for a great **#earlconf***

*Dang, **#earlconf** is over again? I really need to bribe my boss to send me there one year…*

Harold Selman also wrote a great round up of the conference **on his Linkedin page.**

From delegate feedback:

94% of delegates who replied to the feedback said they got what they wanted from EARL London, we were also left some lovely comments:

My whole data science team attended the conference and gained from both a wider learning experience and team building. The team got to discuss potential new projects out of the office environment, which really enhanced our brainstorming! This will add huge value to our future business benefit.

Fantastic conference. EARL never lets me down in providing an insightful, applicable and fun experience to learn from other companies on how they apply R in their enterprises.

EARL is a must for any organisation that wants to understand how R and technologies around R can enable them to achieve better results.

As an R user I've been lucky enough to attend EARL now for the last three years. Each time I come away with a fizz of ideas and energised for my year ahead

And not forgetting our **mention in the Telegraph!**

A thank you

To all of our speakers, sponsors, delegates and the whole Mango team for another fantastic EARL. We will get started on EARL 2020 in early January – so keep an eye out for the call for abstracts then! If you have any comments or ideas either **tweet us** or **drop us an email.**

To **leave a comment** for the author, please follow the link and comment on their blog: **RBlog – Mango Solutions**.

R-bloggers.com offers **daily e-mail updates** about **R** news and tutorials about **learning R** and many other topics. **Click here if you're looking to post or find an R/data-science job**.

Want to share your content on R-bloggers? **click here** if you have a blog, or **here** if you don't.

This posting includes an audio/video/photo media file: Download Now

How to build analytics platforms – Part 5: Data visualization and reliable results

Posted: 25 Sep 2019 05:15 AM PDT

[This article was first published on **R-Bloggers – eoda GmbH**, and kindly contributed to **R-bloggers**]. (You can report issue about the content on this page **here**)

Want to share your content on R-bloggers? **click here** if you have a blog, or **here** if you don't.

What does a modern analytics platform need to offer companies real added value?

Contradiction and connection at the same time: Information content vs. simple presentation

A meaningful result or the answer to important questions is not everything. Making data and results comprehensible is just as important for companies as target-oriented analysis[1]. The more complex a situation is, the more difficult it becomes to present the necessary information in an **easy way**. But that's exactly what platforms have to face these days: To present the abundance of data and the results in an understandable way. This is a difficult undertaking. Especially when distributing information to non-experts, it is important to make the gained knowledge comprehensible. The density of information and a presentation that is not overburdened is an art. Everyone who has ever created an infographic knows that. However, this is exactly what an **analytics platform** must deliver.

It is particularly important that no false impression is created even with a simple representation: Disproportionate scale ratios, incomprehensible axis labels, a poorly distinguishable choice of color and many other factors can lead to misunderstandings, an unintentional distortion of results and incorrect interpretations. In this context, other aspects need to be considered – we will discuss this in the next paragraph.

Visualization of resilient results, acceptance and trust

If data is visualized, the next question arises: How can companies rely on the results? Often human competence has to be relied upon. If the results of the analyses are sent back and forth between different applications, Excel sheets or content systems, edited, returned, released and re-sent, there must inevitably be additional control mechanisms to ensure the accuracy of the data. Accordingly, there is an additional effort.

One consideration of how this could be done would be to have the environment in which the various analyses take place provide a visualization of the results at the same time. If the platform traces back each data point to the actual data sources, the people who present the results can not only act faster, they can also provide information to others without additional reinsurance. In this way, the analysis environment would also significantly improve the flow of information within the organization. For example, the results of analysis projects could be displayed in real time on screens at different locations. With reference to the previous section, different screen sizes and ratios also need to be considered. Small screens compress, wide screens lengthen. This can lead to false impressions. This shows which factors must be considered with a genuine scalable solution.

CONCLUSION: If a data science platform should be used company-wide, it must not be measured exclusively by the use of analyses. Especially the „non-scientific“ aspects are of high relevance, so that all participants in the company can profit from the generated benefits.

Check out the previous articles about our analytics platform YUNA.

- **UI & Teams**
- **Intelligent user and role concept**
- **Individualized workflows and dashboards**
- **Database scalability and business models**

To **leave a comment** for the author, please follow the link and comment on their blog: **R-Bloggers – eoda GmbH**.

R-bloggers.com offers **daily e-mail updates** about **R** news and tutorials about **learning R** and many other topics. **Click here if you're looking to post or find an R/data-science job**.

Want to share your content on R-bloggers? **click here** if you have a blog, or **here** if you don't.

This posting includes an audio/video/photo media file: Download Now

Time series forecasting with random forest

Posted: 25 Sep 2019 04:15 AM PDT

[This article was first published on **r-bloggers – STATWORX**, and kindly contributed to **R-bloggers**]. (You can report issue about the content on this page **here**)

Want to share your content on R-bloggers? **click here** if you have a blog, or **here** if you don't.

Benjamin Franklin said that only two things are certain in life: death and taxes. That explains why my colleagues at **STATWORX** were less than excited when they told me about their plans for the weekend a few weeks back: doing their income tax declaration. Man, I thought, that sucks, I'd rather spend this time outdoors. And then an idea was born.

What could taxes and the outdoors possibly have in common? Well, I asked myself: can we predict tax revenue using random forest? (wildly creative, I know). When dealing with tax revenue, we enter the realm of time series, ruled by fantastic beasts like ARIMA, VAR, STLM, and others. These are tried and proven methods, so why use random forests?

Well, you and I may both agree that random forest is one of the most awesome algorithms around: it's simple, flexible, and powerful. So much so, that Wyner et al. (2015) call it *the* ‚off-the-shelf' tool for most data science applications. Long story short, it's one of those algorithms that just works (if you want to know exactly how then check out this excellent **post** by my colleague Andre).

Random forest is a hammer, but is time series data a nail?

You probably used random forest for regression and classification before, but time series forecasting? Hold up you're going to say; time series data is special! And you're right. When it comes to data that has a time dimension, applying machine learning (ML) methods becomes a little tricky.

How come? Well, random forests, like most ML methods, have no awareness of time. On the contrary, they take observations to be independent and identically distributed. This assumption is obviously violated in time series data which is characterized by serial dependence.

What's more, random forests or decision tree based methods are unable to predict a trend, i.e., they do not extrapolate. To understand why, recall that trees operate by if-then rules that recursively split the input space. Thus, they're unable to predict values that fall outside the range of values of the target in the training set.

So, should we go back to ARIMA? Not just yet! With a few tricks, we *can* do time series forecasting with random forests. All it takes is a little pre- and (post-)processing. This blog post will show you how you can harness random forests for forecasting!

Let it be said that there are different ways to go about this. Here's how we are going to pull it off: We'll raid the time series econometrics toolbox for some old but gold techniques – differencing and statistical transformations. These are cornerstones of ARIMA modeling, but who says we can't use them for random forests as well?

To stick with the topic, we'll use a time series from the German Statistical Office on the German wage and income tax revenue from 1999 – 2018 (after tax redistribution). You can download the data **here**. Let's do it!

Getting ready for machine learning or what's in a time series anyway?

Essentially, a (univariate) time series is a vector of values indexed by time. In order to make it ‚learnable' we need to do some pre-processing. This can include some or all of the following:

- Statistical transformations (Box-Cox transform, log transform, etc.)

- Detrending (differencing, STL, SEATS, etc.)
- Time Delay Embedding (more on this below)
- Feature engineering (lags, rolling statistics, Fourier terms, time dummies, etc.)

For brevity and clarity, we'll focus on steps one to three in this post.

Ok, let's structure this a bit: in order to use random forest for time series data we do TDE: transform, difference and embed. Let's fire up R and load the required packages plus our data.

```
# load the packages
suppressPackageStartupMessages(require(tidyverse))
suppressPackageStartupMessages(require(tsibble))
suppressPackageStartupMessages(require(randomForest))
suppressPackageStartupMessages(require(forecast))

# specify the csv file (your path here)
file <- ".../tax.csv"

# read in the csv file
tax_tbl <- readr::read_delim(
  file = file,
  delim = ";",
  col_names = c("Year", "Type", month.abb),
  skip = 1,
  col_types = "iciiiiiiiiiiii",
  na = c("...")
) %>%
  select(-Type) %>%
  gather(Date, Value, -Year) %>%
  unite("Date", c(Date, Year), sep = " ") %>%
  mutate(
    Date = Date %>%
      lubridate::parse_date_time("m y") %>%
      yearmonth()
  ) %>%
  drop_na() %>%
```

```
  as_tsibble(index = "Date") %>%
  filter(Date <= "2018-12-01")

# convert to ts format
tax_ts <- as.ts(tax_tbl)
```

Before we dive into the analysis, let's quickly check for implicit and explicit missings in the data. The tsibble package has some handy functions to do just that:

```
# implicit missings
has_gaps(tax_tbl)

# explicit missings
colSums(is.na(tax_tbl[, "Value"]))
```

Nope, looks good! So what kind of time series are we dealing with?

```
# visualize
plot_org <- tax_tbl %>%
  ggplot(aes(Date, Value / 1000)) + # to get the axis on a more manageable scale
  geom_line() +
  theme_minimal() +
  labs(title = "German Wage and Income Taxes 1999 - 2018", x = "Year", y = "Euros")
```

Differencing can make all the difference

If you've worked with classical time series models before, you likely stumbled across the concept of differencing. The reason for this is that classical time series models require the data to be stationary.

Stationarity means that the mean and variance of the series is finite and does not change over time. Thus, it implies some stability in the statistical properties of the time series. As we can see in the plot, our time series is far from it! There is an upward trend as well as a distinct seasonal pattern in the series.

How is it done? Here, we simply take the first differences of the data, i.e., the difference between consecutive observations . This also works with a seasonal lag , which amounts to taking the difference between an observation and a previous observation from the same season, e.g., November this year and November last year.

How do these two concepts – differencing and stationarity – relate? You probably already know or guessed it: differencing is one way to make non-stationary time series data stationary. That's nice to know, but right now we care more about the fact that differencing removes changes in the level of a series and, with it, the trend. Just what we need for our random forest!

How is it done? Here, we simply take the first differences of the data, i.e., the difference between consecutive observations . This also works with a seasonal lag , which amounts to taking the difference between an observation and a previous observation from the same season, e.g., November this year and November last year.

Whereas differencing can stabilize the mean of a time series, a Box-Cox or log transformation can stabilize the variance. The family of Box-Cox transformations revolves around the parameter lambda:

When lambda is zero, the Box-Cox transformation amounts to taking logs. We choose this value to make the back-transformation of our forecasts straightforward. But don't hesitate to experiment with different values of lambda or estimate the ‚best' value with the help of the forecast package.

```
# pretend we're in December 2017 and have to forecast the next twelve months
tax_ts_org <- window(tax_ts, end = c(2017, 12))

# estimate the required order of differencing
n_diffs <- nsdiffs(tax_ts_org)

# log transform and difference the data
tax_ts_trf <- tax_ts_org %>%
  log() %>%
  diff(n_diffs)

# check out the difference! (pun)
plot_trf <- tax_ts_trf %>%
  autoplot() +
  xlab("Year") +
  ylab("Euros") +
  ggtitle("German Wage and Income Taxes 1999 - 2018") +
  theme_minimal()

gridExtra::grid.arrange(plot_org, plot_trf)
```

Let's sum up what we've done so far: we first took logs of the data to stabilize the variance. Then, we differenced the data once to make it stationary in the mean. Together, these rather simple transformations took us from non-stationary to stationary.

What's next? Well, now we use the data thus transformed to train our random forest and to make forecasts. Once we obtain the forecasts, we reverse the transformations to get them on the original scale of the data.

Just one more step before we get to the modeling part: we still only have a vector. How do we cast this data in a shape, that an ML algorithm can handle?

Enter the matrix: Time Delay Embedding

To feed our random forest the transformed data, we need to turn what is essentially a vector into a matrix, i.e., a structure that an ML algorithm can work with. For this, we make use of a concept called time delay embedding.

Time delay embedding represents a time series in a Euclidean space with the embedding dimension . To do this in R, use the base function `embed()`. All you have to do is plug in the time series object and set the embedding dimension as one greater than the desired number of lags.

```
lag_order <- 6 # the desired number of lags (six months)
horizon <- 12 # the forecast horizon (twelve months)

tax_ts_mbd <- embed(tax_ts_trf, lag_order + 1) # embedding magic!
```

When you check out the `tax_ts_mbd` object, you'll see that you get a matrix where the dependent variable in the first column is regressed on its lags in the remaining columns:

Time delay embedding allows us to use any linear or non-linear regression method on time series data, be it random forest, gradient boosting, support vector machines, etc. I decided to go with a lag of six months, but you can play around with other lags. Moreover, the forecast horizon is twelve as we're forecasting the tax revenue for the year 2018.

When it comes to forecasting, I'm pretty direct

In this post, we make use of the direct forecasting strategy. That means that we estimate separate models , one for each forecast horizon. In other words, we train a separate model for each time distance in the data. For an awesome tutorial on how this works check out this **post**.

The direct forecasting strategy is less efficient than the recursive forecasting strategy, which estimates only one model and, as the name suggests, re-uses it times. Recursive, in this case, means that we feed back each forecast as input back to the model to get the next forecast.

Despite this drawback, the direct strategy has two key advantages: First, it does not suffer from an accumulation of forecast errors, and second, it makes it straightforward to include exogenous predictors.

How to implement the direct forecasting strategy is nicely demonstrated in the before-mentioned post, so I don't want rehash it here. If you're short on time, the **tl;dr** is this: we use the direct forecasting strategy to generate multi-step ahead forecasts. This entails training a model for each forecast horizon by progressively reshaping the training data to reflect the time distance between observations.

```
y_train <- tax_ts_mbd[, 1] # the target
X_train <- tax_ts_mbd[, -1] # everything but the target

y_test <- window(tax_ts, start = c(2018, 1), end = c(2018, 12)) # the year 2018
X_test <- tax_ts_mbd[nrow(tax_ts_mbd), c(1:lag_order)] # the test set consisting
# of the six most recent values (we have six lags) of the training set. It's the
# same for all models.
```

If you followed me here, kudos, we're almost done! For now, we get to the fun part: letting our random forest loose on this data. We train the model in a loop, where each iteration fits one model, one for each forecast horizon .

The random forest forecast: things are looking good

Below I'm using the random forest straight out of the box, not even bothering tuning it (a topic to which I'd like to dedicate a post in the future). It may seem lazy (and probably is), but I stripped the process down to its bare bones in the hope of showing most clearly what is going on here.

```
forecasts_rf <- numeric(horizon)

for (i in 1:horizon){
  # set seed
  set.seed(2019)

  # fit the model
  fit_rf <- randomForest(X_train, y_train)
```

```
  # predict using the test set
  forecasts_rf[i] <- predict(fit_rf, X_test)

  # here is where we repeatedly reshape the training data to reflect the time distance
  # corresponding to the current forecast horizon.
  y_train <- y_train[-1]

  X_train <- X_train[-nrow(X_train), ]
}
```

Alright, the loop's done. We just trained twelve models and got twelve forecasts. Since we transformed our time series before training, we need to transform the forecasts back.

Back to the former or how we get forecasts on the original scale

As we took the log transform earlier, the back-transform is rather straightforward. We roll back the process from the inside out, i.e., we first reverse the differencing and then the log transform. We do this by exponentiating the cumulative sum of our transformed forecasts and multiplying the result with the last observation of our time series. In other words, we calculate:

```
# calculate the exp term
exp_term <- exp(cumsum(forecasts_rf))

# extract the last observation from the time series (y_t)
last_observation <- as.vector(tail(tax_ts_org, 1))

# calculate the final predictions
backtransformed_forecasts <- last_observation * exp_term

# convert to ts format
y_pred <- ts(
  backtransformed_forecasts,
  start = c(2018, 1),
  frequency = 12
)
```

```
# add the forecasts to the original tibble
tax_tbl <- tax_tbl %>%
  mutate(Forecast = c(rep(NA, length(tax_ts_org)), y_pred))

# visualize the forecasts
plot_fc <- tax_tbl %>%
  ggplot(aes(x = Date)) +
  geom_line(aes(y = Value / 1000)) +
  geom_line(aes(y = Forecast / 1000), color = "blue") +
  theme_minimal() +
  labs(
    title = "Forecast of the German Wage and Income Tax for the Year 2018",
    x = "Year",
    y = "Euros"
  )

accuracy(y_pred, y_test)
```

	ME	RMSE	MAE	MPE	MAPE	ACF1	Theil's U
Test set	198307.5	352789.9	238652.6	2.273785	2.607773	0.257256	0.0695694

It looks like our forecast is pretty good! We achieved a MAPE of 2.6 percent. But since one should never rely on accuracy metrics alone, let's quickly calculate a simple benchmark like the seasonal naive model. That's a low hurdle to pass, so if our model doesn't beat it, in the bin it goes.

```
benchmark <- forecast(snaive(tax_ts_org), h = horizon)

tax_ts %>%
  autoplot() +
  autolayer(benchmark, PI = FALSE)

accuracy(benchmark, y_test)
```

	ME	RMSE	MAE	MPE	MAPE	MASE	ACF1	Theil's U
Training set	216938.7	470874.8	388618.5	2.852533	6.648711	1.000000	0.6303982	NA
Test set	485006.8	495096.8	485006.8	5.507943	5.507943	1.248028	0.1382461	0.0926723

The error metrics are much higher here, so it's safe to say our random forest did a good job.

Where do we go from here? Well, we haven't tried it yet, but we may further improve our forecasts with some hyperparameter tuning. Also, just between the two of us, maybe random forest is pretty good, but not the best model for the job. We could try others or an ensemble of models.

If you take away one thing from this post today, let it be this: We can do effective time series forecasting with machine learning without whipping out big guns like recurrent neural networks. All it takes is a little pre- and post-processing. So why not include random forest in your arsenal the next time you do forecasting (or procrastinate on doing your taxes)? 🙂

References

- Wyner, Abraham J., et al. „Explaining the success of adaboost and random forests as interpolating classifiers." The Journal of Machine Learning Research 18.1 (2017): 1558-1590.

Über den Autor

Manuel Tilgner

I am a data scientist at **STATWORX**, and I enjoy making data make sense. Why? Because there's something magical about turning a jumble of numbers into insights. In my free time, I love wandering through the forest or playing in the local big band.

ABOUT US

STATWORX
is a consulting company for data science, statistics, machine learning and artificial intelligence located in Frankfurt, Zurich and Vienna. Sign up for our NEWSLETTER and receive reads and treats from the world of data science and AI. If you have questions or suggestions, please write us an e-mail addressed to blog(at)statworx.com.

Sign Up Now!

Sign Up Now!

Der Beitrag **Time series forecasting with random forest** erschien zuerst auf **STATWORX**.

To **leave a comment** for the author, please follow the link and comment on their blog: **r-bloggers – STATWORX**.

R-bloggers.com offers **daily e-mail updates** about **R** news and tutorials about **learning R** and many other topics. **Click here if you're looking to post or find an R/data-science job**.

Want to share your content on R-bloggers? **click here** if you have a blog, or **here** if you don't.

This posting includes an audio/video/photo media file: Download Now

You are subscribed to email updates from R-bloggers.
To stop receiving these emails, you may unsubscribe now.
Google, 1600 Amphitheatre Parkway, Mountain View, CA 94043, United States

Email delivery powered by Google